Lego™ Mindstorms™ Robots as a Platform for Teaching Reinforcement Learning

Peter Vamplew
School of Computing
University of Tasmania
Sandy Bay, Tasmania, Australia
Peter.Vamplew@utas.edu.au

ABSTRACT

This paper examines the suitability of Lego™ Mindstorms™ robotic kits as a platform for teaching the concepts of reinforcement learning. The reinforcement learning algorithm Sarsa was implemented on board an autonomous Mindstorms™ robot, and applied to two learning tasks. The reasons behind the differing results obtained on these two tasks are discussed, and several issues related to the suitability of Mindstorms™ as a platform for teaching or experimentation with reinforcement learning are identified.

1. INTRODUCTION

Over recent years the increasing availability of low-cost programmable robotics systems has lead to a growth in the use of robots within an educational context. At primary and secondary school level, events such as RoboCupJunior have been seen as a means of developing technical literacy amongst students by providing them with an exciting, highly motivating learning environment [1] It has also been suggested that these motivational benefits apply at the level of undergraduate teaching with [2, 3] amongst others promoting the use of robots in introductory computer science and artificial intelligence courses. The research described in this paper was motivated by a proposal to incorporate robotics within the teaching of an intelligent agents course within the School of Computing at the University of Tasmania.

Reinforcement learning would appear to be an aspect of artificial intelligence for which this teaching methodology is particularly appropriate. This form of machine learning is gathering an increasing amount of attention amongst artificial intelligence researchers, and as such should be considered for inclusion in the curriculum of undergraduate artificial intelligence units. The reinforcement learning paradigm involves an autonomous agent which learns through interaction with its environment the optimal outputs to produce in response to observed inputs in order to maximise the overall reinforcement received. This approach is well suited to tasks where a goal can be specified for the system, but where expert knowledge about the means of achieving that goal does not exist.

An autonomous robot is a natural example of a situation to which reinforcement learning is applicable. The robot gathers input about the environment via its sensors, and can interact with the environment via its manipulators. The task desired of the robot is defined in terms of a reinforcement signal, and the robot aims to choose its actions in such a way as to maximise that reinforcement. Control systems for autonomous robots have been one of the main applications of reinforcement learning (for example [4] and [5]), and so this would appear to be an extremely appropriate example to use in teaching this style of machine learning. In particular the use of a robot situated within a real, physical location emphasises the role of the environment within reinforcement learning much more clearly than would be the case within a simulation or more abstract application. However [3] states that "learning techniques such as neural networks and reinforcement learning are too complex to implement directly on the LEGO hardware". This paper aims to show that by carefully selecting the task and robot configuration, it is in fact possible to successfully implement and demonstrate reinforcement learning using a low-cost robotics system.

2. LEGO™ MINDSTORMS™ AND LEGOS

Clearly one of the main restrictions on the adoption of educational robotics is the cost involved in providing enough robotics hardware for students to have sufficient hands-on experience. The Mindstorms™ robots manufactured by Lego™ provide a possible platform for use in such a course. These robots are relatively low-cost, programmable, and due to being constructed from Lego blocks have a flexible physical structure which can be adapted to a wide-range of tasks.

Unfortunately the standard software bundled with a Mindstorms™ robot has several major restrictions from the point of view of implementing machine learning algorithms. Most important amongst these limitations are the extremely small number of variables available, and the lack of support for floating point arithmetic. However alternative programming environments providing enhanced functionality such as Not Quite C [6] and LegOS [7] have been developed by Mindstorms™ enthusiasts. For the purposes of this research LegOS was selected as the implementation environment. LegOS programs are written in C or C++, and cross-compiled under a Linux environment before being downloaded to the RCX controller on the Mindstorms™ robot. It should be noted that during development of the software for this project numerous discrepancies were found between the LegOS libraries and the corresponding documentation. Whilst this is perhaps to be expected given LegOS's 'alternative' status and on-going development (and is in fact alluded to in

the documentation), it would nevertheless prove troublesome were students asked to develop code in this environment.

3. THE LEARNING TASKS AND ALGORITHM

Two different tasks were chosen to test the LegOS implementation of reinforcement learning. These problems had different problem characteristics as well as requiring different physical configurations of the robot. In both cases known heuristic algorithms were available to allow comparison with the behaviour achieved by the learning system.

As stated earlier, the task facing a reinforcement learning system is to find an appropriate mapping from input to output values so as to maximise the reinforcement received by the learning agent. In this case the inputs to the system are measurements of the environment made by the robot's sensors, whilst the outputs are used to control the robot's effectors. With regards to the Mindstorms™ robots the available inputs are light and contact sensors, whilst the outputs are generally used to drive the robot's motors. Using only the components from a single Robotics Invention System™ kit, the robot is restricted to three inputs and two outputs. (There are in fact three outputs, but only two motors).

The most flexible control of the robot would be obtained by mapping each output directly to a control value for a single motor. However most current reinforcement learning algorithms have difficulties in dealing with outputs of continuous nature (although progress has been made in this area - for example see [8]). The alternative is to define a small number of discrete actions which are available to be selected by the control system. For any given state information, the control system will estimate the value of executing each of these actions in that situation and select the action with the highest expected return.

The software used in this research implements the Sarsa [9] algorithm which is based on the concept of temporal difference learning [10]. Sarsa learns through experience the value associated with selecting a particular action in a particular state. After an action is selected and executed, its estimated value is updated on the basis of any immediate reward and the estimated value of the action which will be selected for the subsequent state. The Sarsa algorithm is given below; Q(s,a) is the estimated value of taking action a from state s, r is the immediate reward received after taking an action, α is a learning-rate parameter, and γ is a discounting factor.

```
Initialise Q(s,a) with random values
For each learning episode
   Observe initial state s
   Select an action a to perform
   Repeat
      Execute a, observe r and the new state s'
      Select action a' from s'
      Q(s,a) = Q(s,a) + α(r + γQ(s',a') - Q(s,a))
      s = s'; a = a'
   Until end of episode
```

The action selection process is based on the current Q values. In order to balance the need to make exploratory actions to learn about the problem space with the need to exploit what has already been learnt about the environment, an ε-greedy strategy is used in which the action is selected greedily with probability 1-ε, and randomly with probability ε.

The simplest implementation of Sarsa is the tabular form in which the Q values are stored in a table with a cell for each state-action pair. This does not scale well to problems with a large number of states (for example in the line-following task described in Section 3.2 the table would require 256 x 256 x 3 = 196608 cells), and so the Q values are often represented instead using some form of function approximation. For this work linear approximators were used to represent the values of state-action pairs – this function-fitting system would be too simple for many tasks, but from analysis of the two sample tasks used in this study it could be seen that a more complex approximation function was not necessary. Linear methods have previously been successfully applied within a reinforcement learning context by several authors, including [11] and [12].

For the tasks used in these experiments it was found that it was necessary to insert a delay loop in the implementation of the Sarsa algorithm, as otherwise the amount of movement executed by the robot between action selections was so small that the vast majority of actions did not result in any changes in either the input state or the reinforcement signal. The need for this delay indicates that the processing power of the CPU would be sufficient to scale up to more sophisticated forms of function approximation (such as small neural networks) should this be required for more complex learning tasks.

3.1. THE WALKING TASK

The first task used in this experiment involved the robot learning to perform a walking action. The robot structure used for this task had four legs, but with both legs on the same side of the body being driven by a single shared motor, as shown in Figure 1. Notice the contact sensor positioned directly above the motor (this is duplicated on the far side of the robot), and also the downward facing light sensor at the front of the robot.

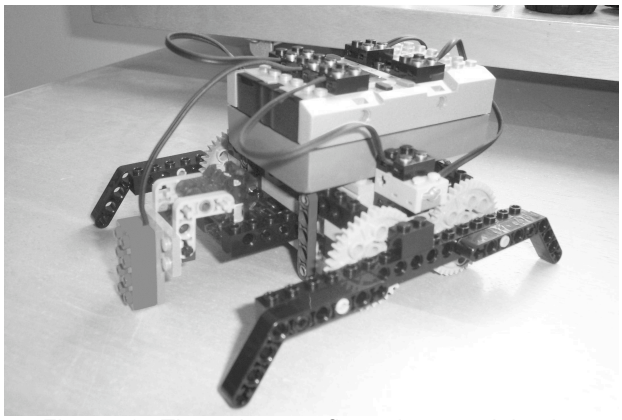

Figure 1: The robot configuration used in the walking task experiments.

In order for such a robot to walk (or more precisely, shuffle) it is necessary for the two pairs of legs to move out of phase with each other. If both motors are run simultaneously the effect is that the legs remain fixed, whilst the body rotates around the motors. A walking action can be achieved by a heuristic algorithm which introduces a pause in the activation of each motor, timed to create an asymmetry in the movement of the legs. In this algorithm, there is no input from the external environment. Instead the pausing of the motors is determined by sensing the positioning of the robot's legs via the contact sensors which are pressed when each leg is raised to its highest position relative to its motor. Varying the length of the pause gives rise to gaits of differing appearance and effectiveness, as investigated by [13].

This task was chosen to investigate whether a reinforcement learning system could learn to emulate or improve on any of the walking actions produced by the heuristic walking algorithm. The inputs to the TD(0) algorithm used were similar to those used by the heuristic algorithm. For each motor a count was maintained of the number of time-steps for which that motor had been active since its corresponding contact sensor had last been activated. The inputs to the linear approximators consisted of this value for each motor, as well as the difference between the values for the two motors. All inputs were scaled to the range –0.5 to 0.5. Essentially this provides the robot with a simple model of the relative positioning of its legs.

There were three possible actions to select from – both motors on full, left motor on full and right motor off, and left motor off and right motor on full. Therefore the system contained a linear approximator for each of these actions, which were initially set to small random weights.

The desired behaviour for the robot was to walk forward as quickly as possible. This was measured by placing the robot on a sheet of paper printed with a smooth greyscale gradient from black to white. After each time-step the brightness of the surface below the robot was sensed via a single light sensor, and this was compared to the corresponding value for the previous time-step to determine whether the robot had moved forward. Movements in a forward direction were rewarded with positive reinforcement, whilst backwards moves were penalised with negative reinforcement, and failure to move received zero reinforcement.

3.2. THE LINE-FOLLOWING TASK

The second task required a wheeled robot (based on the Roverbot design in the Lego Constructopedia, with the addition of two side-by-side light sensors mounted at the front of the chassis, as shown in Figure 2) to follow a black line marked on a white surface. The line was sufficiently wide for both sensors to be positioned over it at the same time if the robot was oriented directly along the line, and it formed a track consisting of a circuit with a mixture of 10 left and right hand bends of varying sharpness.

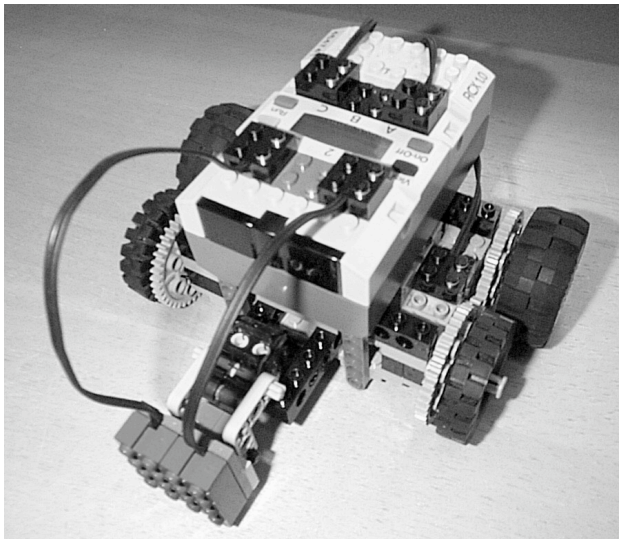

Figure 2: The robot configuration used in the line-following experiments.

A heuristic algorithm was constructed and tested to ensure that the task of following the line was possible. This algorithm used the ratio between the values of the left and right light sensors to select between three pre-defined behaviours (move forward with both motors on equally, turn left by setting the left motor to run backwards slowly whilst the right motor ran forwards at a higher speed, and turn right). A robot following this algorithm completed a lap of the circuit after the execution of around 2300 actions. Note that a delay loop was built into this algorithm to ensure that it was selecting actions at the same rate as the reinforcement learning system, so as to allow for valid comparisons to be made between the performance of the two control systems.

The inputs to the linear approximators used in the reinforcement learning system for this task were the values of the light sensors (scaled to a range of –0.5 to 0.5 by calibration relative to values obtained from a light and dark surface at the start of the training process). These approximators were used to select between the same three actions (forward, turn left, turn right) used by the heuristic line-following algorithm.

There was no direct way to measure the progress of the robot around the circuit, and so no reinforcement function which directly corresponded to the desired behaviour could be constructed. Instead two alternative reinforcement signals were used which attempted to approximately correspond to the desired behaviour. The details of these, and their effect, are discussed in the results section.

4. EXPERIMENTAL METHOD

Several trials were run for each of the learning tasks, using different initial random weights. Each trial consisted of many episodes. At the beginning of each episode the robot was placed in a starting position within the learning environment, and a button on the RCX block (the main body of the robot) was pressed to indicate whether this was a training or testing episode.

During training episodes, 95 % of the actions were selected based on the expected values indicated by the linear approximators, whilst the remaining 5% of the actions were selected randomly, and the weights of the linear approximators were updated using the SARSA algorithm, with a learning rate of 0.1 and a discounting factor of 0.9. During test episodes no weight updates were performed, and all actions were selected greedily according to the current policy. Episodes were ended manually via an RCX button whenever the robot left the surface prepared for the task, or when the task was successfully completed.

There was a large amount of manual involvement required to start and stop each training and testing episode, and in resetting the robot to its starting position prior to commencing each episode. Therefore it was not practical to run a large number of trials and analyse the results statistically. Given that the intention of this paper is to examine the suitability of the Mindstorms™ robot as a teaching platform rather than to compare alternative learning algorithms this was not seen as a major issue. Hence the results presented will be primarily qualitative rather than quantitative in nature.

5. RESULTS AND DISCUSSION

5.1. THE WALKING TASK

The performance of the reinforcement learning algorithm on the walking task was consistently poor. The learning system always converged to a solution where both motors were constantly turned on. This can give rise to a moderately effective gait, but only when the legs are not initially symmetrically positioned. The robot was always started from a symmetrical configuration as this was its natural resting position when the motors were inactive. During training the occasional randomly selected action was sufficient to eventually break the symmetry of the legs and so the robot would make some limited progress along the measuring gradient. However during testing all actions were selected greedily, and the learning system did not prove capable of learning to produce this symmetry-breaking behaviour of its own accord, and therefore no forward progress was achieved during test episodes.

During these experiments several factors were observed which contributed to this failure. Firstly the values of the light sensor were noted to be extremely noisy. Variations of up to 10% of the overall range of the light sensor were observed whilst the sensor was in a static position over a surface of constant intensity. This was further exacerbated by variations in the height or orientation of the light sensor relative to the surface as the robot moved. As a result the reinforcement signal received by the learning system was extremely noisy.

A second problem which was observed was slippage in the positioning of the robot. Whenever one of the motors was deactivated, the weight of the robot would cause the gears attached to that motor to turn back to a position where the robot body was as low as possible relative to the motor. This slippage could be slowed by applying the motor's brake, but this merely reduced the speed of the movement rather than eliminating it completely and had the side-effect of rapidly depleting the batteries. The result of this unintended motion of the robot was a growing disparity between the robot's model of the position of its legs and their actual position over the course of each training episode. This was a major problem as the model of the robot's position was the only input provided to the learning system.

5.2. THE LINE-FOLLOWING TASK

After some experimentation with the reinforcement signal, the reinforcement learning system was much more successful on the line-following task than on the walking task.

In the initial trials, the reinforcement signal used rewarded the robot with positive reinforcement for any action which led to the robot remaining on the track (measured by applying a threshold to the summed value of the light sensors). As the actions available did not provide an option for staying still, this was expected to lead to the robot moving forward along the path and eventually traversing the circuit. However the learning algorithm discovered that alternating turning left and right allowed the robot to reverse slowly in a straight line, and hence maximal reinforcement could be achieved by travelling along a straight section of line at the beginning of the track, and then reversing back along that same section of track.

To overcome this, the reinforcement function was modified so as to only reward the robot on time-steps on which it moved forward whilst remaining on the track. This required the reinforcement function to take into account which action had been performed (as there was no other means of determining whether the robot had moved forward) Therefore the reinforcement function was no longer being determined strictly on the basis of the environment which violates some of the principles of reinforcement learning. Unfortunately this could not be avoided given the limited sensory capabilities of the robot. (In principle the track itself could have been marked out with a greyscale gradient, but given the problems in sensing such a gradient which were observed during the walking task, this was not expected to be successful).

Using this revised reinforcement function, the learning system proved capable of learning to execute the task correctly in most trials. After around 20 training episodes (which took about 20 minutes to carry out), the robot successfully navigated an entire lap of the circuit in both training and testing modes. The number of actions required for this lap in testing mode was on the order of 1700, which was approximately 25% faster than the speed achieved by the heuristic algorithm. Table 1 shows an example of the improvement in the robot's performance over a series of training episodes. Note that to reduce the overall amount of time involved in training the robot, testing was only carried out when there had been signs of improved performance during the previous training episode.

	Training		Testing	
Episode number	Time steps	Bends	Time steps	Bends
0			71	0
1	95	0	86	0
2	89	0		
3	99	0		
4	297	1	175	0
5	297	1		
6	330	1		
7	254	1		
8	514	1		
9	474	1		
10	307	1		
11	349	1		
12	2075	Lap	263	1.5
13	452	1		
14	341	1		
15	280	1		
16	395	1		
17	2084	Lap	1710	Lap

Table 1: Sample results from training on the line-following problem. Results are reported in terms of the number of decision-making cycles for which the robot remained on the track, and the number of bends of the track successfully negotiated during this time.

Even with the revised reinforcement function occasional unsuccessful trials were still noted, in which the robot learnt an interesting form of aberrant behaviour. Early in training the robot is likely to lose the path, and then makes essentially random moves until it either rediscovers the path or leaves the training area (at which point the episode is ended). In some cases the robot made sufficient consecutive turns in the same direction to regain the path but facing in the opposite direction. If these episodes were allowed to continue, the behaviour which resulted was that the robot would follow the path until a bend was encountered, at which point it would turn 180° and head back in the opposite direction. This pattern of behaviour could be repeated each time a bend was reached whilst still receiving a high level of reinforcement.

6. CONCLUSION

The failure of the learning system on the walking task can be attributed to two factors – the noise in the light sensor which resulted in a very noisy reinforcement signal, and the robot's inability to accurately sense its own configuration. For this task there was no input from the external environment, and so all state information was derived from the robot's model of its own current position. The combination of the inability to directly monitor the leg position, and the large amount of slippage which occurred when a motor was set to inactive, meant that the state information provided to the learning algorithm was extremely inaccurate. One possible solution would be to make use of rotation sensors (not supplied in the standard Mindstorms™ kit) to provide direct monitoring of the motor positions.

In contrast the input information for the line-following task was derived entirely from the external environment, with no need to monitor the robot's internal state. The noisiness of the light sensors also posed fewer difficulties in this task, as the information being derived from those sensors was less sensitive to small fluctuations over time. The primary issue affecting success on this task was the nature of the reinforcement signal. It was observed that using a reinforcement signal which does not directly correspond to the desired behaviour can lead to the robot learning aberrant behaviours which receive high reinforcement but which are not desirable. Far from being a problem, this occurrence is actually beneficial to the teaching of reinforcement learning, as it emphasises the key role played by the reinforcement function in this style of learning. Observations of similar unexpected behaviour have been made by other authors in reinforcement-style learning tasks - for example [14] reported this in the context of evolutionary computation.

Of more concern from an educational perspective, the Mindstorms™ robot's limited capacity for sensing its environment made construction of a suitable reinforcement function for the line-following task difficult. If the robot could measure greyscale values more reliably, it may have proved possible to mark the path with a greyscale gradient so that direction and distance of movement along the path could be sensed. This would have enabled the construction of a reinforcement signal derived entirely from the environment, with no need to explicitly consider the action being performed by the robot, which would have been more in line with the reinforcement learning paradigm.

The learning algorithm used in these experiments was a relatively simple form of reinforcement learning, using the single-step form of the temporal difference algorithm (TD(0)) in conjunction with a simplistic function approximator (the linear approximator). However the processing power of the RCX was more than sufficient to cope with the demands of this learning algorithm, and the software used could easily be extended to implement more complex forms of reinforcement learning such as TD(λ) or the use of small neural networks as approximators.

This study has shown that the combination of the Mindstorms™ robot and the LegOS programming environment is sufficient for student experimentation with reinforcement learning as long as the limitations of

the robots are taken into account in the choice of the task to be learnt. The key characteristics of the task are that the robot's behaviour should be determined by the state of the environment rather than by the robot's own configuration, and that it should be possible to define an appropriate reinforcement signal which corresponds as closely as possible to the desired behaviour of the robot whilst still being capable of being derived by the robot on the basis of its limited sensors.

7. ACKNOWLEDGMENTS

I wish to acknowledge the assistance of the Centre for Computational Intelligence at De Montfort University, Leicester, UK for the use of their facilities in conducting this research.

REFERENCES

[1] Sklar, E., Eguchi, A. & Johnson, J. (2002). RoboCupJunior: Learning with educational robotics. In *Proceedings of RoboCup-2002: Robot Soccer World Cup VI*.

[2] Kumar, D. & Meeden, L. (1998). A Robot Laboratory for Teaching Artificial Intelligence. In Joyce, D. (Ed.). *Proceedings of the Twenty-ninth Technical Symposium on Computer Science Education (SIGCSE-98)*, ACM Press.

[3] Sklar, E., & Parsons, S. (2002). RoboCupJunior: A vehicle for enhancing technical literacy. In *Proceedings of the AAAI-02 Mobile Robot Workshop*.

[4] Mahadevan, S. & Connell, J. (1992). Automatic programming of behavior-based robots using reinforcement learning, *Artificial Intelligence*, Vol. 55, 311-365.

[5] Asada, M., Noda, S., Tawaratsumida, S. & Hosoda, A. (1996). Purposive Behavior Acquisition for a Real Robot by Vision-Based Reinforcement Learning, *Machine Learning*, Vol. 23, 279-303.

[6] Baum, D. (n.d.) Retrieved February 10 2003 from http://www.baumfamily.org/nqc/

[7] Noga, M. (n.d.) Retrieved February 8 2003 from http://www.noga.de/legOS/

[8] Gaskett, C., Wettergreen, D., & Zelinsky, A. (1999). Q-Learning in Continuous State and Action Spaces. In *Proceedings of the 12th Australian Joint Conference on Artificial Intelligence*, Springer-Verlag.

[9] Rummery, G. A., & Niranjan, M. (1994). *On-line q-learning using connectionist systems* (No. CUED/F-INFENG/TR 166): Cambridge University Engineering Department.

[10] Sutton, R.S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning, Vol. 3*, 9-44.

[11] Sutton R.S. (1996). Generalisation in reinforcement learning: Successful examples using sparse coarse coding. In Touretzky D.S., Mozer M.C., & Hasselmo M.E. (Eds.). *Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference* (1038-1044). Cambridge, MA: The MIT Press.

[12] Stone P., Sutton R.S., & Singh S. (2001). "Reinforcement learning for 3 vs. 2 Keepaway", in Stone P, Balch T. and Kreatzschmarr G. (Eds.), *RoboCup-2000: Robot Soccer World Cup IV*, Berlin:Springer-Verlag.

[13] Graves, A.R. & Bierton, R.A. (2000). *Investigation of Inter-Step Pause Values for a Simple Legged Robot*. Unpublished internal report of the Centre for Computational Intelligence, De Montfort University.

[14] Sims, K. (1994). Evolving Virtual Creatures. *Computer Graphics (Siggraph '94 Proceedings),*15-22